Cyril Canonizado, David Lu, Eric Wong, Olivia Zhang

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

cyril.canonizado@gmail.com (Cereal Box)

david.lu349@gmail.com (Iron Man)

ewong18195@gmail.com (Edwin Vancleef)

olivia.ny.zhang@gmail.com (Recursive Yemechan)

Assignment 3: VPN

October 15, 2014

Requirements And Setup

Our program requires that the operating system running the program has Java SE Runtime Environment 7 or above installed. Our program will be contained within a .jar file and therefore will not require any installation. To run, double click on the .jar file. At the top, two tabs will be displayed; a client tab and a server tab.

Sending A Message

To connect to a server, click on the client tab. Then enter hostname and port into text fields and click connect. If the server is up, a connection will be established. Once established, a message will be displayed announcing the connection in the text area of the lower part of the program. Then you may enter a secret value in which you and the participant in your conversation share. Then to send a message, enter the message into the text field next to the "send message button" and click "send message" once done.
To host a server, click on the server tab. Then enter the port you wish your client to connect to. If the port is available, the program will attempt to wait for a connection request. Once successfully connected, a message will be displayed announcing the connection in the text area of the lower part of the program. To send a message, enter a message into the text field next to the "send message" button and click "send message" once done.

Our Key Encryption is done with RSA. For this, we used Java's security library, which offers functionality for key generation and en/de-cryption with various cryptography algorithms such as RSA.

For Key Establishment, we have chosen to implement a Diffie-Hellman protocol. We chose Diffie-Hellman because it's unfeasible to break when intercepting only the values that are shared publicly. Diffie-Hellman allows for a secure communication channel that preserves confidentiality and integrity of data to those with authenticated keys. Unfortunately, when we tried to combine our Diffie-Hellman implementation with our RSA implementation, our programs does not work properly. We were unable to fix this merging problem in time. However, we do have Diffie-Hellman and RSA implemented properly on separate branches. However, although we have written code for Diffie-Hellman and have pushed it in various branches in our Github repo (such as merge-DH and authentication), the final code in master does not contain it as we had issues integrating it with the rest of our code.

In our planned Diffie-Hellman approach, the sender and the client will enter a number in the secret value field. This will act as their secret key for the DH protocol. When the sender

sends a message, the DH protocol takes into effect by first generating a modulus and a generator value, and calculating the value of: generator^secret key mod modulus for both the sender and the receiver. Then, both values are sent to each other. Our program then calculates the shared secret value by calculating the value of the sent number raised to the power of the respective secret keys, then taking the modulo of that with the modulus value. If the resulting values are the same for both the sender and receiver, then the sender is authenticated and their encrypted message is received by the receiver.

In our final design, we managed to implement a VPN capable of establishing a connection between a client and a server. It is capable of sending encrypted message as well as decrypting the message upon receipt. Work has been done to implement a Diffie-Hellman key authentication protocol; unfortunately, we have not been able to integrate it with the rest of the VPN. As such, the final VPN does not do authentication.

VPN As Real World Product

If we were to develop the VPN as a real-life product, we would most likely opt for a large key size (like 2048 bits). In our implementation, we opted for a 1024-bit size for the key generator. This size was chosen arbitrarily. With a larger key size, the VPN could be more secure. Because of that, we would use RSA-2048 as our main encryption algorithm. As for key authentication, we feel as though that Diffie-Hellman is still a very good option, and would likely reuse it.

Our VPN was written entirely in Java and the user interface was created with the help of the Java GUI-builder, Swing. The program size was 830 SLOC.

Link to Project Github Repository:
https://github.com/David8472/EECE412Project